Creating and Deploying an ERC-20 Token with Python + Solidity on Kali Linux

This guide will walk you step-by-step through creating and deploying your own ERC-20 token using Solidity and Python on Kali Linux. We will use a Python virtual environment, install Solidity, write a custom smart contract, compile it, and deploy it to the Sepolia Ethereum test network.

PHASE 1 — Environment Setup on Kali

```
1. Create a project folder:
    mkdir mycoin_project
    cd mycoin_project

2. Create & activate a virtual environment:
    python3 -m venv venv
    source venv/bin/activate

3. Install the required Python packages:
    pip install py-solc-x web3 python-dotenv

4. Install a Solidity compiler inside venv:
    python3
    >>> from solcx import install_solc
    >>> install_solc("0.8.17")
    >>> exit()
```

PHASE 2 — Write Your Solidity Token Contract

Create a file named MyCoin.sol with the following content:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract MyCoin {
    string public name = "My Custom Coin";
    string public symbol = "MCC";
    uint8 public decimals = 18;
    uint256 public totalSupply;

    mapping(address => uint256) public balanceOf;
    mapping(address => mapping(address => uint256)) public allowance;

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);

    constructor(uint256 _initialSupply) {
        totalSupply = _initialSupply * (10 ** uint256(decimals));
        balanceOf[msg.sender] = totalSupply;
    }

    function transfer(address _to, uint256 _value) public returns (bool) {
        require(balanceOf[msg.sender] >= _value, "Not enough balance");
        balanceOf[msg.sender] -= _value;
        balanceOf[_to] += _value;
        emit Transfer(msg.sender, _to, _value);
        return true;
    }

    function approve(address _spender, uint256 _value) public returns (bool) {
        allowance[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value);
        return true;
    }

    function transferFrom(address _from, address _to, uint256 _value) public returns (bool) {
        require(balanceOf[_from] >= _value, "Not enough balance");
        require(allowance[_from][msg.sender] >= _value, "Allowance exceeded");
        balanceOf[_from] -= _value;
        allowance[_from][msg.sender] -= _value;
        balanceOf[_to] += _value;
        emit Transfer(_from, _to, _value);
        return true;
    }
}
```

PHASE 3 — Prepare Python Deployment Script

1. Create a `.env` file and add:

```
    PRIVATE_KEY=your_wallet_private_key_here
    INFURA_URL=https://sepolia.infura.io/v3/YOUR_INFURA_PROJECT_ID
    ACCOUNT_ADDRESS=your_wallet_address_here
```

2. Create a file deploy.py with the following content:

```
import json
import os
from web3 import Web3
from solcx import compile_standard, install_solc
from dotenv import load_dotenv

load_dotenv()

PRIVATE_KEY = os.getenv("PRIVATE_KEY")
INFURA_URL = os.getenv("INFURA_URL")
ACCOUNT_ADDRESS = os.getenv("ACCOUNT_ADDRESS")

w3 = Web3(Web3.HTTPProvider(INFURA_URL))
chain_id = 11155111

with open("MyCoin.sol", "r") as file:
    mycoin_source = file.read()

install_solc("0.8.17")
compiled_sol = compile_standard(
    {
        "language": "Solidity",
        "sources": {"MyCoin.sol": {"content": mycoin_source}},
        "settings": {"outputSelection": {"*": {"*": ["abi", "metadata", "evm.bytecode", "evm.sourceMa
    },
    solc_version="0.8.17",
)

bytecode = compiled_sol["contracts"]["MyCoin.sol"]["MyCoin"]["evm"]["bytecode"]["object"]
abi = compiled_sol["contracts"]["MyCoin.sol"]["MyCoin"]["abi"]

MyCoin = w3.eth.contract(abi=abi, bytecode=bytecode)
nonce = w3.eth.get_transaction_count(ACCOUNT_ADDRESS)

transaction = MyCoin.constructor(1000000).build_transaction(
    {
        "chainId": chain_id,
        "from": ACCOUNT_ADDRESS,
        "nonce": nonce,
        "gas": 2000000,
        "gasPrice": w3.to_wei("20", "gwei"),
    }
)

signed_txn = w3.eth.account.sign_transaction(transaction, private_key=PRIVATE_KEY)
tx_hash = w3.eth.send_raw_transaction(signed_txn.rawTransaction)
print(f"Deploying contract... TX Hash: {tx_hash.hex()}")

tx_receipt = w3.eth.wait_for_transaction_receipt(tx_hash)
print(f"■ Contract deployed at address: {tx_receipt.contractAddress}")
```

PHASE 4 — Run Deployment

1. Get free Sepolia test ETH from https://sepoliafaucet.com/
2. Run the deployment script:

```
    python3 deploy.py
```

```
If successful, you'll see:
Deploying contract... TX Hash: 0x....
■ Contract deployed at address: 0xYourContractAddress
```

PHASE 5 — Verify & Use Your Coin

- Open https://sepolia.etherscan.io/ and search for your contract address. - You'll see your coin's details and can interact with it. - Anyone can add your token to MetaMask using the same contract address and symbol. This process gives you a fully functional ERC-20 token on Ethereum's Sepolia testnet, deployed from Kali Linux using Python and Solidity.